I will like to implement a Database for this site using REPLIT. I need a plan or PRD to explain REPLIT that this Store list could store, several hundred of stores, I need to be reliable, but at the same time cost effective, ease to integrate into the website and with capabilities to backup the information, restore the information, and to retrieve all the recorded data (including text and images) for backup. In your capacity of Database expert and years of experience in Full-stack development, Can you help me wit this task?

Certainly! Here's a structured plan and PRD (Product Requirement Document) outline for implementing a reliable, cost-effective, and easy-to-integrate database solution for your store list website on Replit.

Product Requirement Document (PRD) for Store List Database on Replit

1. Overview

Create a scalable and reliable database solution to store, retrieve, and backup data for several hundred store info cards, including text data, images, and location maps, with features supporting backup, restore, and efficient data retrieval.

---

2. Goals and Objectives

- Store detailed store info (name, address, contact phone, email, website)
- Store classification tags (type, region, state)
- Store images (logo, map representation)
- Store Google Maps link
- Ensure data reliability and integrity
- Enable easy data backup and restore
- Support efficient data retrieval for display on the website
- Cost-effective usage of resources
- Easy integration with the Replit-hosted website

---

3. Architectural Approach

Database Type:
Use **SQLite** (local, serverless, lightweight) or **Replit DB** (built-in key-value store).
For more scalable needs, consider external cloud options like **Supabase** or **Firebase**, but for simplicity and cost-efficiency, Replit DB is suitable.

Recommended choice:
Replit DB (since it's seamlessly integrated with Replit, simple, and cost-effective)

---

4. Data Model Design

Store Info Card Data Structure:

Field	Type	Description
id	String/UUID	Unique identifier for the store
name	String	Store name
address	String	Physical address
telephone	String	Contact number
email	String	Email address
website	String	Store website URL
tags	Array of Strings	Classification tags (type, region, state)
logoImage	String	URL or storage reference for Logo image
storeImage	String	URL or storage reference for Store image
mapImage	String	URL or storage reference for Map image
googleMapsLink	String	Link to Google Maps

Notes:

- Store images ideally stored on an external storage service (e.g., Replit Files, Cloudinary, etc.), with URLs stored in the DB.
- For local images, Replit Files can be used, but external hosting is more scalable.

---

5. Features & Functionalities

A. Store CRUD operations

- Create: Add new store data
- Read: Retrieve all or specific store data
- Update: Modify existing store info
- Delete: Remove store records

B. Data Backup & Restore

- Export all data into a JSON file or CSV for backup
- Import data from backup files to restore

C. Data Retrieval

- Fetch all store records (with optional filtering by tags, region, or type)
- Paging support for large data sets

D. Media Storage & Retrieval

- Use external storage or Replit Files for images
- Store URLs or paths in the database for access

---

6. Implementation Plan

Step 1: Set up Replit DB in your Replit project

- Use the Replit DB package/library

Step 2: Define data schemas and create CRUD functions in your backend code

- Functions to add, get, update, delete store info

Step 3: Implement media management

- Upload images to Replit Files or external hosting
- Save URLs in the database

Step 4: Backup & restore functions

- Export: Traverse all records, export as JSON/CSV
- Import: Read backup files, populate database

Step 5: Integration with Frontend

- Fetch data via API endpoints or direct DB calls
- Display store info dynamically

---

7. Cost & Reliability Considerations

- **Replit DB** is free and persistent, suitable for several hundred entries.
- Regular backups (manual or scheduled) mitigate data loss.
- For increased reliability, consider external backups (periodic exports).
- Use error handling to ensure stability during CRUD operations.

---

8. Additional Recommendations

- Use UUIDs or incremental IDs for unique store identification.
- Store images externally for scalability.
- Implement filters on the front-end for categories/tags.
- Document all APIs or functions for ease of maintenance.
